RS-232C Serial Interfacing with the NEURON® CHIP

August 1991 LONWORKS™ Engineering Bulletin

Introduction

The NEURON CHIP is a programmable device that includes a rich variety of input/output capabilities. The NEURON CHIP's firmware can configure the 11 I/O pins of the processor in 25 different modes supported by software drivers. Application programs running on the processor can then access this I/O functionality through simple calls to the I/O driver functions. The NEURON CHIP is also a device that can communicate with other NEURON CHIPs over a variety of networking media, such as twisted-pair wiring, radio-frequency (RF), and powerline, using the LONTALK™ protocol. It is thus an ideal device to implement intelligent distributed control (IDC) applications.

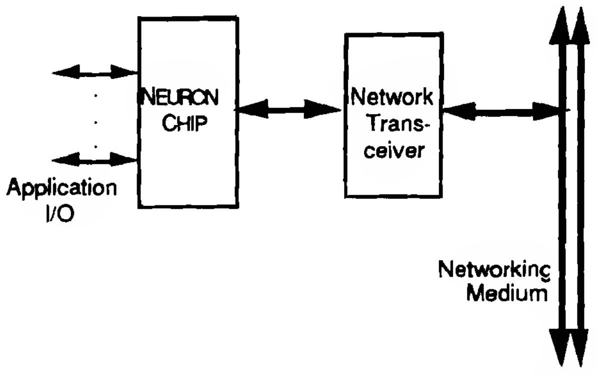

Figure 1. Architecture of a NEURON CHIP-based node.

The LONWORKS™ network uses a multi-drop concurrent-access architecture, so that multiple nodes can communicate in a peer-to-peer fashion. The RS-485 standard supports such a multi-drop architecture, allowing any node to communicate with any other node on the network, with up to 32 nodes per physical channel. On the other hand, the RS-232C standard is a point-to-point architecture, which allows only two devices to communicate with each other. The standard was originally designed for communications between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) such as modems. However, it has been widely applied in recent years to other point-to-point communications needs.

This engineering bulletin describes the implementation of an RS-232C asynchronous serial interface that enables a NEURON CHIP to communicate with another device that employs the RS-232C standard. This could be, for example, a CRT terminal, a printer, a card reader, or a modem. The same NEURON CHIP can also be a part of a network communicating with the LONTALK protocol, forming, in effect, a gateway between an RS-232C link and a LONWORKS network.

Asynchronous Data Format

The NEURON CHIP receives and transmits serial data using eight-bit character frames, with one start bit and one stop bit. The RS-232C standard defines two voltage levels. The negative voltage corresponds to logic level 1 and the positive voltage to logic level 0. When the line is idle, it is at logic level 1. A character frame begins with a start bit, which holds the line at 0 for one bit time. Then the eight data bits are transmitted with the least significant bit first. Finally, the line returns to 1 for at least one bit time, forming the stop bit. A new character frame may start at any time after the end of the stop bit. This asynchronous data format is commonly used with the RS-232C standard interface, although strictly speaking, it is not a part of the standard. It is termed asynchronous since it is not necessary to share a clock between the transmitting and receiving devices. Both devices can use independent local clocks running at the same nominal frequency. Actual synchronization is on a character-by-character basis using the start and stop bits.

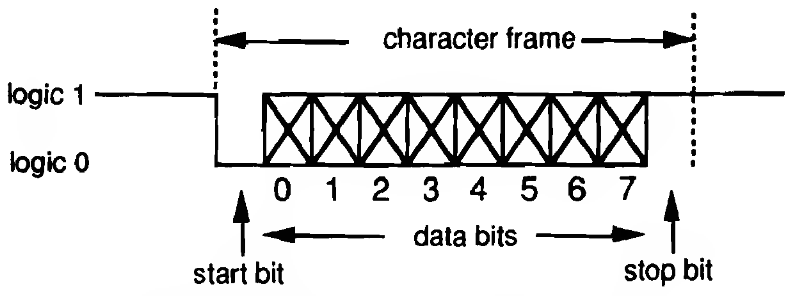

Figure 2. Asynchronous data format.

Hardware Considerations

The NEURON CHIP supports this asynchronous serial data format using the serial device type. The serial output device is implemented on pin IO.10 and the serial input device on pin IO.8. The nine remaining I/O pins can be used for other I/O functions. The I/O pins have TTL input levels and standard CMOS output levels. Devices such as the Motorola MC 145407 may be used to convert these levels to and from RS-232C voltage levels. Figure 3 shows a typical schematic for a bi-directional RS-232C interface for a NEURON CHIP configured as Data Terminal Equipment (DTE). The interface chip chosen is a 5-volt-only driver/receiver that uses an on-chip charge pump to generate the RS-232C voltage levels with the help of four external capacitors.

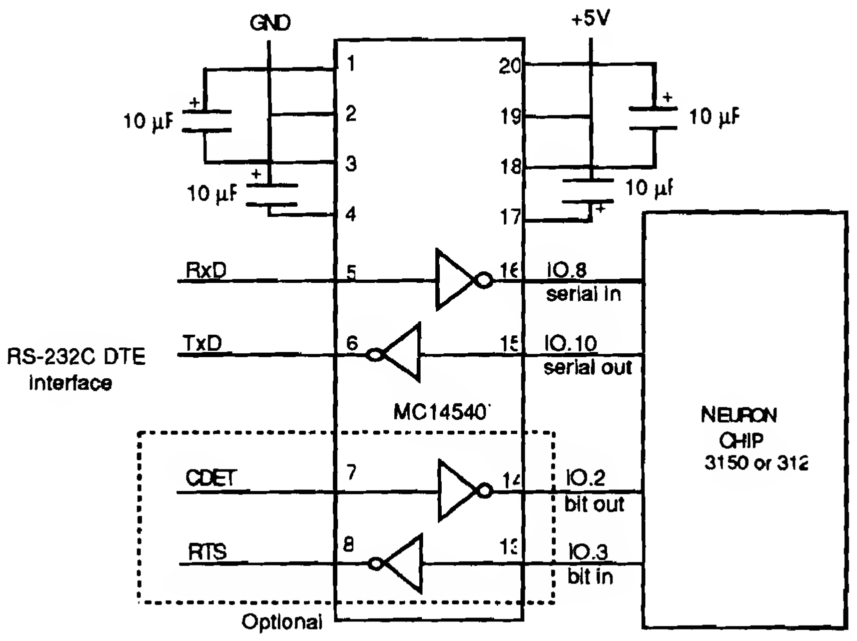

Figure 3. Typical RS-232C interface circuit.

If additional modem control lines such as CDET, DSR, DTR, CTS and RTS are required, then any of the other NEURON CHIP I/O lines may be configured as bit input or output lines under control of the application software. Not all of these signals are active high. The application software should check which sense is valid and handle each one of them appropriately. For full details on the modem control lines, see the Electronics Industries Association (EIA) RS-232C Standard document.

Software Considerations

The designer uses statements in the NEURON C programming language to declare and activate serial devices. This provides a high-level interface that frees the application designer from considerations of bit timing, data framing, and character assembly and disassembly.

Serial Output

To declare a serial output device, a statement of the form should be used:

```
IO_10 output serial baud (constant) io_object_name;
```

For example, the following statement declares the device named `CRT_screen` as a serial output device operating at 300 baud:

```
IO_10 output serial baud( 300 ) CRT_screen;
```

To output data to the serial device, use a statement of the form:

```
io_out( io_object_name, buffer_pointer, character_count );
```

For example, the following statement sends the twelve ASCII characters `"Hello, world"` in serial format on pin IO.10:

```
io_out( CRT_screen, "Hello, world", 12 );
```

The parameters to the `io_out` function call for a serial output device are:

`io_object_name`	A name declared as a serial output object
`buffer_pointer`	A parameter of type `(char *)` pointing to an array of characters
`character_count`	A parameter of type `(unsigned int)`

The `io_out` function call suspends execution of the application program until all the characters have been transmitted on the output pin. For example, transmitting 60 characters at 300 baud will suspend the application for 60 * (1 + 8 + 1) / 300 = 2 seconds. During serial I/O, the network and media access processors on the NEURON CHIP continue to execute the network protocol, but the application processor does not execute other tasks to handle any generated events. Application designers should take this into account.

The allowable values for the constant expression used to specify the baud rate are 300, 1200, 2400, and 4800, which refer to the serial bit rate at a NEURON CHIP input clock rate of 10 MHz. If other input clock rates are used, refer to Table 1.1.

Input Clock (MHz)	*baud(300)*	*baud(1200)*	*baud(2400)*	*baud(4800)*
10.0	300	1200	2400	4800
5.0	150	600	1200	2400
2.5	75	300	600	1200
1.25	37.5	150	300	600
0.625	18.75	75	150	300

Table 1: Serial bit rates for different NEURON CHIP input clock rates.

Serial Input

To declare a serial input device, a statement of the form should be used:

```
IO_8 input serial baud (constant) io_object_name;
```

For example, the following statement declares the device named `keyboard` as a serial input device, operating at 300 baud:

```
IO_8 input serial baud( 300 ) keyboard;
```

To input data from the serial device, a statement of the form should be used:

```
num_chars_received =
   io_in( io_object_name, buffer_pointer, max_character_count );
```

For example, the following statement causes the NEURON CHIP to wait for up to twelve serial characters to be received in serial format on pin IO_8:

```
num_chars_received = io_in( keyboard, input_buffer, 12 );
```

The parameters to the `io_in` function call for a serial input device are:

io_object_name	A name declared as a serial input object
buffer_pointer	A parameter of type `(char *)` pointing to an array of characters to accept the received data

max_character_count A parameter of type `(unsigned int)`

The `io_in` function returns a value of type `(unsigned int)`, indicating the number of characters actually received. Suitable declarations for the above example are:

```
char input_buffer[ 12 ];
unsigned int num_chars_received;
```

Note that in the C language, an object of type `(char [])` is automatically promoted to type `(char *)` in the context of a function call. It is the designer's responsibility to ensure that the input buffer for serial input is large enough to contain the number of characters requested. If the `io_in` call specifies a maximum character count that exceeds the size of the input buffer, unpredictable behavior will result.

The `io_in` function suspends application processing until it is complete. This occurs at the first of the following:

1. The number of characters specified in the third parameter of the function call have all been received
2. The line has been continuously at the idle level for 250 msec (independent of NEURON CHIP input clock), or
3. A framing error has occurred – an expected start bit or stop bit has the wrong polarity.

In all cases, the returned value of the function is the number of characters actually received and stored in the buffer.

Synchronizing with an External Serial Device

Because the serial devices are implemented using software-driven serial I/O, there are a few restrictions in the use of serial input:

- The application program must be waiting at an `io_in` function call when the start bit of the character is received. If the call to `io_in` is made after the beginning of the start bit of a character, an immediate framing error is likely. If the call to `io_in` is made after the end of a character, then that character will be totally missed.
- If the start bit of the first character is delayed more than 250 msec after calling the `io_in` function, then the call will time out and no characters will be returned. A time-out will also occur if there is a pause of more than 250 msec between the end of one character and the beginning of the next. In this case, the call will return the characters received up until the time-out.

Solutions to these limitations depend on the device that is generating the input characters. If the input device is not an operator typing at a keyboard, but rather another processor, then some form of hardware handshaking can be implemented. For example, the NEURON CHIP can activate a logical output line (using `output bit` mode) that indicates to the device generating characters that the NEURON CHIP is ready to receive data. The NEURON CHIP then immediately enters the `io_in` call for the serial input device. The external device, when it receives the CTS indication, waits until the NEURON CHIP is in the *`io_in`* call, and then transmits its characters. If the number of characters transmitted is fixed in advance, then the *`io_in`* call can specify this number of characters. Otherwise, it can wait 250 msec for the time-out, or the input device can generate a ***break*** condition on the line at the end of the data, causing a framing error and termination of input.

Using Serial I/O to Interface with a CRT Terminal

If the device generating characters is an operator typing at a keyboard, then there are user interface issues to be considered. Normally, an operator will type with unpredictable pauses between characters. These pauses may exceed the 250 msec time-out limit. The serial *`io_in`* function does not check for any input terminators such as the ASCII carriage return character, which is conventionally used to indicate the end of user input. It also does not echo input characters, nor does it recognize any input line editing characters such as back-spaces, as would be expected by a user typing at an interactive terminal device.

If this kind of functionality is required, it must be implemented by the application code calling `io_in` and `io_out` to perform the basic I/O. The following code shows how some of this interactive terminal capabilities might be implemented. The function `get_line` reads characters one at a time from the serial input device, processing some control characters and echoing printable characters to the serial output device. A null-terminated string is returned in `input_buf`.

Following the function `get_line` is a demonstration task that calls `get_line` to read a line from the keyboard and then echoes it back to the screen.

```
// Sample RS-232 driver for interactive serial ASCII keyboard device

#include <control.h>                           // define 'watchdog_update'
IO_8  input serial baud( 2400 ) char_in;      // serial input device
IO_10 output serial baud( 2400 ) char_out;    // serial output device

char input_buf[ 120 ];                         // input buffer for get_line

unsigned get_line ( void ) {
         // function waits for input line, returns number of characters
received

   char * input_buf_ptr;       // next place to store received character
   char * end_of_buf_ptr;      // last byte in buffer
   char this_char;             // character being processed
   unsigned num_chars;

   input_buf_ptr = input_buf;                  // initialize buffer pointers
   end_of_buf_ptr = input_buf + sizeof( input_buf ) - 1;

   while( input_buf_ptr < end_of_buf_ptr ) { // don't read past end of buffer

      do {
         num_chars = io_in( char_in, input_buf_ptr, 1 ); // read one character
         watchdog_update( );                // avoid watchdog timeouts
       } while( !num_chars );               // if error or time_out, try again

      this_char = *input_buf_ptr & 0x7F;       // discard any parity bit
      if( this_char == '\r' ) break;           // exit loop if a carriage return
      if( ( this_char == '\b' ) || ( this_char == 0x7F ) ) {
                                                         // backspace or rubout
         if( input_buf_ptr > input_buf ) {               // buffer is not empty
            input_buf_ptr--;                             // forget last character
            io_out( char_out, "\b \b", 3 );              // erase character on screen
         }
         continue;
      }
      if( this_char < ' ' ) continue;      // ignore other control characters
      *input_buf_ptr = this_char;          // store this character
      io_out( char_out, input_buf_ptr++, 1 );
                                     // echo this character, incr pointer
   }  // end while
   io_out( char_out, "\r\n", 2 );             // send CR LF at end of line
   *input_buf_ptr = '\0';                     // null terminate string
   return( unsigned )( input_buf_ptr - input_buf );
}                                             // return number of characters

// Demonstration task for the get_line() function

when( reset ) {
   unsigned num_chars;

   for( ;; ) {                      // do forever
      num_chars = get_line( );      // read a line
      if( num_chars ) {
         io_out( char_out, output_buf, num_chars );   // echo line
         io_out( char_out, "\r\n", 2 );               // send CR LF
      }
   }
}
```

Disclaimer

Echelon Corporation assumes no responsibility for any errors contained herein.
No part of this document may be reproduced, translated, or transmitted in any form without permission from Echelon.

Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
105 Heath Street
London NW3 6SS
England
Telephone (071) 431-1600
Fax (071) 794-0532
International Telephone + 44 71 431-1600
International Fax + 44 71 794-0532

Echelon Japan K.K.
AIOS Gotanda Building #808
10-7, Higashi-Gotanda 1-chome,
Shinagawa-ku, Tokyo 141, Japan
Telephone (03) 3440-8638
Fax (03) 3440-8639

Part Number 005-0008-01